Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Jere Rahikainen

Ohjelmiston laadunvarmistuksen kehittäminen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

10.5.2020

Tekijä Otsikko Sivumäärä Aika	Jere Rahikainen Ohjelmiston laadunvarmistuksen kehittäminen 32 sivua 10.5.2020
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Ohjelmistotuotanto
Ohjaajat	Lehtori Jorma Räty Toimitusjohtaja Janne Heinonen

Ohjelmiston laadunvarmistuksen kehittämisellä tarkoitetaan tässä työssä automaatiotestauksen mahdollistamista sekä sille tuotettua ohjeistusta. Myös työnhallinnan työkalujen käyttö on osa laadunvarmistamista.

Työn tavoitteena oli toteuttaa automaatiotestauksen manuaalinen työputki. Työpöydällä oli automaatiotestausympäristön pystytys sekä integraatio työnhallinnan työkalun ja automaatiotestauksen välille. Myös ohjeistus automaatiotestausympäristön ylläpidosta ja kehityksen yhtenäisistä tavoista oli toteutettava. Automaatiotestauksen kohteena oli teollisuusvalvomo Ignitioniin pohjautuva web-sovellus.

Automaatiotestausympäristö toteutettiin PyCharm-kehitysympäristössä, Robot Frameworkilla ja sen lisäosa SeleniumLibrarylla. Työnhallintajärjestelmänä toimi Jira ja automaatiotestausympäristön integrointi toteutettiin Jiran lisäosa Xray:n avulla. Versiohallinnan työkaluna toimi Git.

Työ aloitettiin määrittelemällä työssä tarvittavat teknologiat ja suunnittelemalla tarvittavat ohjeistukset. Käytännön töistä aloitettiin ensimmäiseksi automaatiotestausympäristön pystytys asentamalla tarvittavat ohjelmat ja lisäosat. Asennuksien jälkeen tutustuttiin kehityskieleen ja toteutettiin Xray-integraatio. Integraation jälkeen toteutettiin ohjeistukset testauksen eri osa-alueille. Ohjeistuksien perusteella luotiin esimerkkitestejä. Esimerkkitestien yhteydessä käytiin läpi työnhallinnan työkalun mahdollisuuksia.

Verraten lähtötilanteeseen yrityksen valmiudet ehkäistä teknisen velan lisääntymistä on kasvanut. Toteutetut ohjeistukset ja uuden testausympäristön pystytys mahdollistaa testikattavuuden nostamisen. Testien tuottaminen aloitettiin ja Jiran käyttöä myös käyttäjätarinoiden osalta hyödynnettiin. Ohjeistuksia tarkennettiin uuden työntekijän koulutuksen yhteydessä. Kokonaisuutena yrityksen ohjelmistotuotannon kehitysmalli uudistettiin, se sisälsi päivitetyn projektinhallinnan, kokonaan uuden testausympäristön ja automaatiotestauksen.

Avainsanat	Automaatiotestaus, Laadunvarmistus

Author Title Number of Pages Date	Jere Rahikainen Development of Quality Assurance 32 pages 10 May 2020
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Jorma Räty, Senior Lecturer Janne Heinonen, CEO

In this work, the development of quality assurance means enabling automation testing and the instructions produced for it. The use of work management tools is also part of quality assurance.
The aim of the work was to implement a manual automation testing work tube. The desktop had an automation testing environment set up and integration between the work management tool and automation testing. Guidance on the maintenance of an automation testing environment and uniform ways of development also had to be implemented. The object of the automation testing was a web application based on the industrial control room Ignition.

The automation testing environment was implemented in the PyCharm development environment, Robot Framework and its add-on SeleniumLibrary. Jira acted as the work management system and the integration of the automation testing environment was implemented with the help of Jira 's add-on Xray. The version control tool was Git.

The work began by defining the technologies needed in the work and designing the necessary guidelines. The first of the practical work was to set up an automation testing environment by installing the necessary and add-ons. After the installations, the development language was learned and Xray integration was implemented. After the integration, guidelines for the various aspects of testing were implemented. Based on the guidelines, example tests were created. Next to the creation of example tests, the possibilities of the work management tool were reviewed.

Compared to the initial situation, the company's ability to prevent an increase in technical debt has increased. The implemented guidelines and the creation of a new testing environment will make it possible to increase test coverage. Production of the tests began, and the use of Jira was also utilized for user stories. The instructions were specified in connection with the training of a new employee. As a whole, the company’s production development model was revamped and included updated project management, a completely new testing environment, and automation testing.

Keywords	

Sisällys

Lyhenteet

1 Johdanto 1

2 Lähtötilanne 1

3 Teknologiat 2

3.1 Ignition teollisuusvalvomo 2
3.2 Ignition Designer ja Perspective 4
3.3 Robot Framework & SeleniumLibrary 7
3.4 PyCharm 10
3.5 GIT 11
3.6 Jira-ohjelmisto 11
3.7 Xray 14
3.8 Testausmenetelmiä 15
3.8.1 Manuaalinen testaus 15
3.8.2 Automattinen testaus 16

4 Työn kulku 17

5 Yhteenveto 27

Lähteet 29

Lyhenteet

OPC UA	Open Platform Communications (OPC) säätiön kehittämä koneiden välinen viestintäprotokolla teollisuusautomaatioon. (1.)
MQTT	Avoin Open Artwork System Interchange (OASIS)- ja International Organization for Standardization (ISO)-standardien mukainen kevyt verkkoprotokolla, joka kuljettaa tietoja laitteiden välillä. Se on suunniteltu paikkoihin, joissa vaaditaan erittäin kevyttä tiedostokokoa tai verkon kaistanleveys on rajoitettu. (2.)
IoT	Internet of Things eli esineiden internet on joukko järjestelmiä, joissa kahden tai useamman laitteen välillä siirtyy tietoa automaattisesti. Laitteita voi ohjata ja seurata internetin välityksellä. (3.)

1 Johdanto

Insinöörityön tilaaja on Enermix Oy, joka tarjoaa päätuotteena Talotohtori kiinteistöhallintajärjestelmää. Enermix Oy on energiasäästöön, vedensäästöön, talotekniikan automaatioon, energiatehokkuuteen ja pilvipalveluihin keskittynyt asiantuntija- sekä ohjelmistoyritys. Esimerkiksi Enermix Oy tarjoaa älykästä olosuhdepalvelua, jolla voi tasapainottaa kiinteistön sisäilman laatua muun muassa lämpötila-, kosteus- ja CO2-anturien avulla. (4.)

Insinöörityö tehtiin osana Enermix Oy:n ohjelmistokehityksen työmenetelmien uudistusta ja sen pääasiallisena tavoitteena oli luoda hyvät perusteet ohjelmistotuotannon laadunvarmistukselle. Laadunvarmistuksen perusteiksi luotiin yritykselle uusi testausympäristö ja ohjeistus testien luomiseen sekä ylläpitoon. Hyvät perusteet tuovat ohjelmistolle luotettavuutta sekä ohjelmoijalle selkeän suunnitelman, jonka kanssa työskennellä. Ohjelmiston virheet minimoidaan ja ohjelmiston käyttökokemus paranee.

2 Lähtötilanne

Enermix Oy:n Talotohtori-palvelu perustuu Ignition-alustaan, jonka ympärille on luotu lisäominaisuuksia kiinteistöjen hallintaan liittyen. Talotohtori ilmentyy loppukäyttäjälle web-selainsovelluksena sekä mobiilisovelluksena. Talotohtori on kiinteistöhallintajärjestelmä, joka yhdistää esineiden internetin (IoT) ja automaatiojärjestelmät.

Talotohtori-palvelun käyttöliittymän testaaminen oli suoritettu tähän asti manuaalisesti. Manuaalinen testaus suoritettiin siihen varatussa beta-testausympäristössä, jonka jälkeen ohjelmisto siirrettiin tuotantoon. Yksikkötestaustasolla Ignition alusta muodostaa rajoitteita, minkä takia testausautomaatio tehdään pääasiassa järjestelmätasolla käyttöliittymän kautta tapahtuvaksi. Yksikkötestauksella tarkoitetaan ohjelmistokehittäjän tuottaman koodin testaamista (5). Manuaalisessa testauksessa testitapausten toistaminen tasalaatuisesti ja kattavasti on haastavampaa kuin automatisoidussa testauksessa, minkä seurauksena ohjelmistovirheitä on saattanut päästä testauksesta läpi tuotantoon. Manuaalinen testaus on työlästä ja tähän menee liian kauan aikaa eikä kaikkia virheitä aina löydetä.

Ohjelmistokehityksen hallinnassa siirryttiin Scrum-kehitysmallista Kanban-malliin vuoden 2019 lopulla sekä otettiin Jira-ohjelmisto käyttöön. Jira-ohjelmisto on erittäin laaja ja sen ominaisuuksista on alkuvaiheessa otettu käyttöön välttämättömimmät ohjelmistokehityksen jatkuvuuden varmistamiseksi. Kanban on tuotannon ohjaukseen liittyvä ajoitusjärjestelmä, joka on yksi ketterien menetelmien kehitysmalleista. Kanbanin avulla työnkulku on läpinäkyvää ja hidasteet huomataan nopeasti (6).

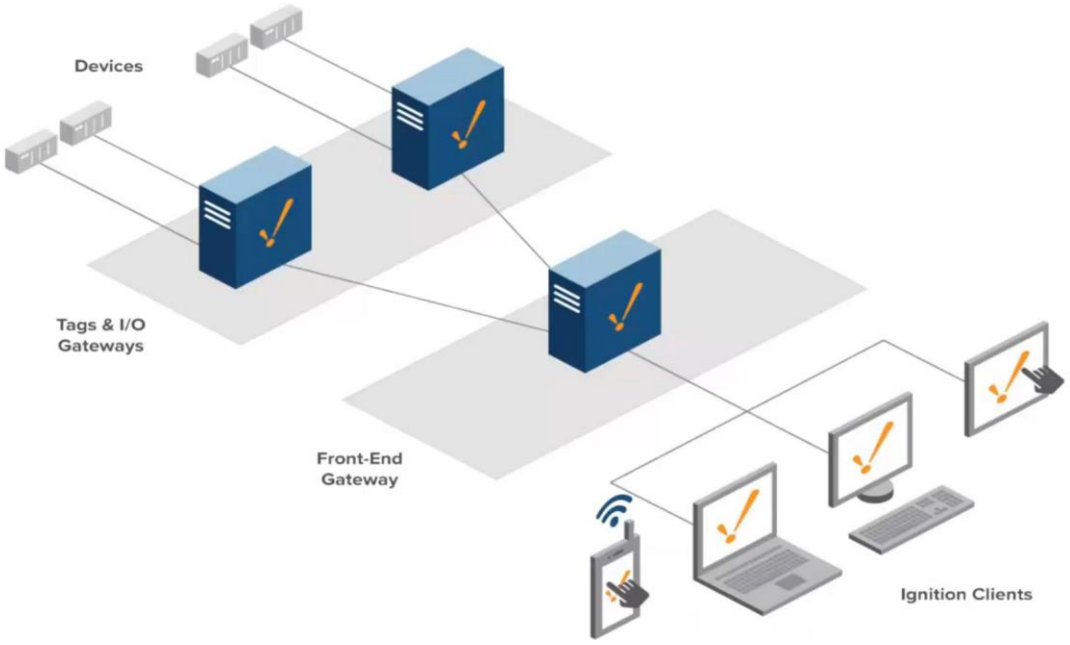
Testauksen automaation suunnittelua tehtiin yhteistyössä Etteplan Oy:n kanssa. Etteplanilla oli kokemusta vastaavanlaisten valvomojärjestelmien testaamisesta ja yhteistyö heidän kanssansa mahdollisti nopean etenemisen tarvittavien testausympäristöjen rakentamisessa. Automaatiotestausympäristön kehityksen työkaluiksi valitsimme PyCharmin ja Robot Frameworkin SeleniumLibrary-lisäkirjastolla. Nämä työkalut yhdistettiin projektihallintajärjestelmä Jiraan ulkoisella Xray-lisäosalla. Versionhallinnan työkaluksi valittiin Git.

3 Teknologiat

Tässä osiossa esitellään projektissa käytettyjä teknologioita ja järjestelmiä.

3.1 Ignition teollisuusvalvomo

Ignition on teollisuuteen suunnattu modulaarinen ohjelmisto, joka tarjoaa kehittäjälle lähes täydellisen kehitysympäristön tietokannasta käyttöliittymän kehitykseen. Alustan on luonut yhdysvaltalainen yritys Inductive automation. Ignition teollisuusvalvomo toimii kaikilla yleisimmillä käyttöjärjestelmillä, se on web-pohjainen, joten sitä on mahdollista käyttää kaikilla laitteilla, joissa on web-selain. Suosituimpia käyttöjärjestelmiä markkinaosuudeltaan viimeisen vuoden ajalta ovat Windows, iOS ja Linux (7). Ignition teollisuusvalvomo koostuu erilaisista moduuleista, jotka vastaavat eri toiminnoista, ne keskustelevat keskenään. (8.) Moduuleja ovat muun muassa Ignition Perspective, SQL Bridge Module, Tag Historian Module, Reportin Module ja Alarm Notification Module (9). Alla on kuva Ignitionin arkkitehtuurista, jossa kuvataan keskikokoisen laitoksen laiteympäristöä.

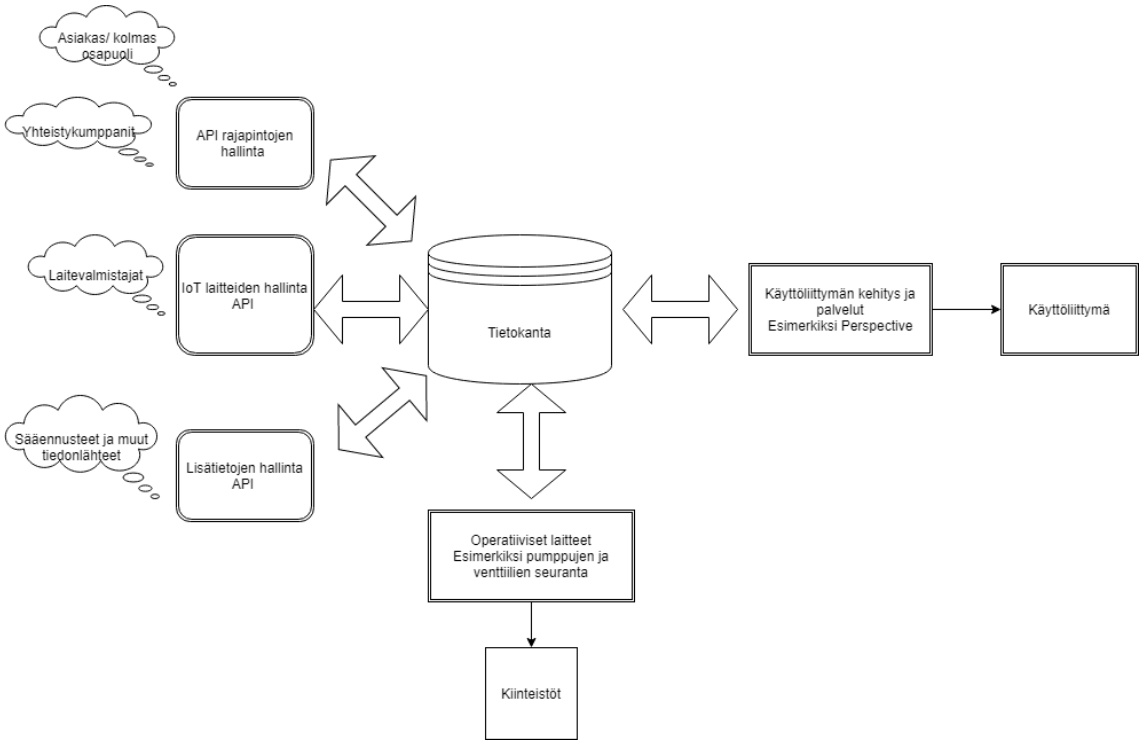


Kuva 1. Ignitionin standardi arkkitehtuurimuoto keskikokoiselle laitokselle. Kuvan oikeassa reunassa kuvataan mobiililaitteet, Ignition Designer ja asiakaspäätteet. Front-End Gateway tarkoittaa käyttöliittymän palvelinta. Tags & I/O Gateways tarkoittaa serveriä, jossa operatiivisiin laitteistoihin luodut yhteydet ovat käynnissä. (10, 2:37.)

Alusta sisältää Talotohtori-palvelun kannalta olennaisia valmiita ominaisuuksia liittyen rakennusautomaatiojärjestelmien liittämiseen eri protokollilla. Talotohtori-palveluun kerätään tietoa rakennusautomaatiojärjestelmistä. Tiedon keräyksessä hyödynnetään useita viestintäprotokollia, esimerkiksi koneiden välistä viestintäprotokollaa (OPC UA) ja kevyttä verkkoprotokollaa (MQTT). Kerätyn tiedon perusteella asetettavat hälytykset (Alarm Notification Module) sekä kerätyn tiedon esittämisen käyttöliittymässä (Tag Historian Module). Tallennetun tiedon perusteella on mahdollista muodostaa raportteja, kuten lämpötilahistoriaan liittyen (Reporting module). (8.)

Ignition alusta ja sen päälle Enermixin kehittämä tietomalli, IoT- ja API-rajapinnat sekä käyttöliittymä mahdollistavat kiinteistöjen IoT- ja automaatiojärjestelmien yhdistämisen samaan järjestelmään. Alla on kuvaus osasta Talotohtorin järjestelmässä käytettävistä moduuleista, jotka osa on Ignitionin tarjoamia ja osa on Enermixin kehittämiä. Kuvasta on jätetty pois muun muassa monitorointijärjestelmät, yhteyksien kuvaukset ja verkkokauppa, jotka eivät ole tarpeellisia esitellä tässä työssä. API-rajapintoihin tehdyt yhteydet ovat Enermixin kehittämiä. Kaikki API-rajapinnoista ja operatiivisilta laitteilta tuleva tieto

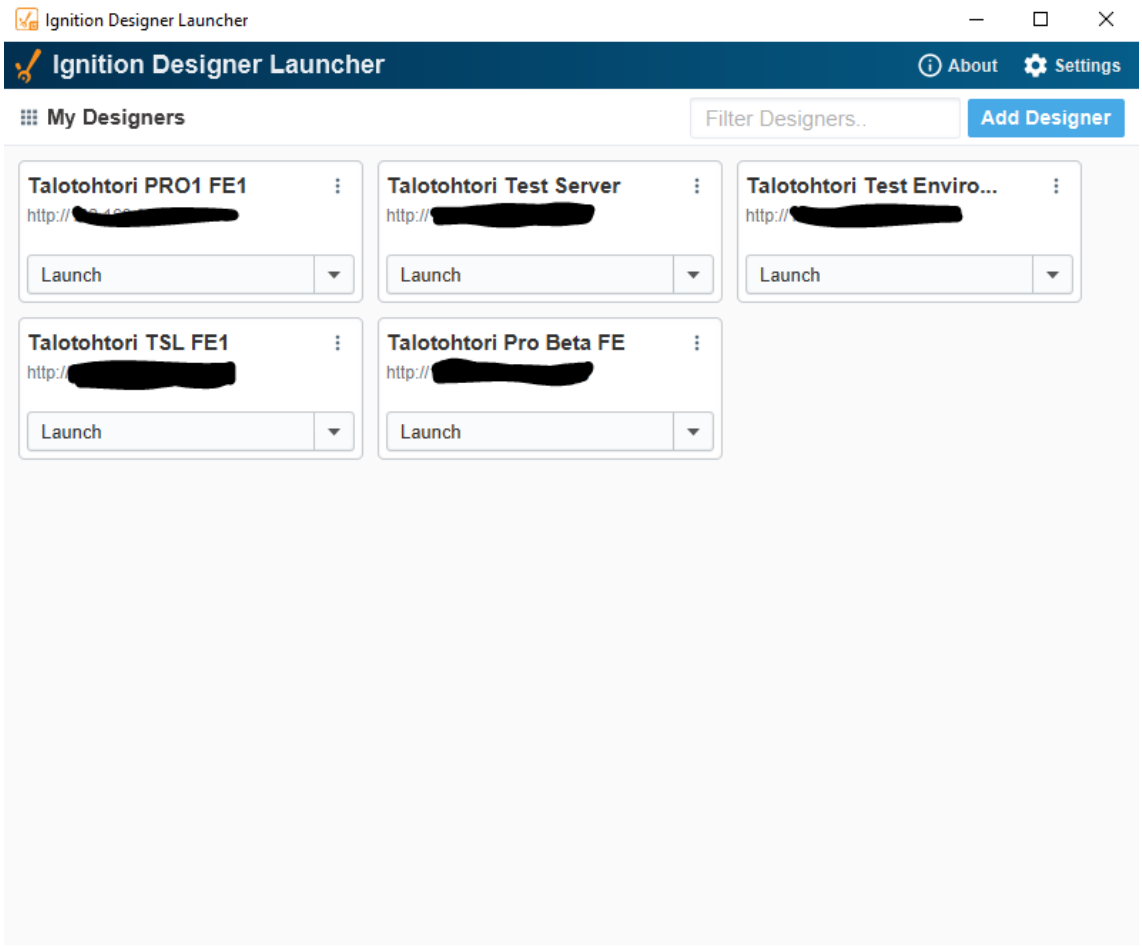
muutetaan vakiomuotoiseen datamalliin ennen kuin se tallennetaan tietokantaan. Vakiotietomallin avulla alan muut toimijat voivat käyttää Talotohtorin tuottamaa tietoa omissa palveluissaan. Operatiiviset laitteet on yhdistetty erilaisilla automaatioprotokollilla kuten esimerkiksi OPC UA ja Drivers for OPC UA moduulien avulla. Käyttöliittymä luodaan Ignition Perspective -moduulilla.



Kuva 2. Osa Talotohtorin järjestelmän moduuleista. Vasemmalla kuvataan rajapintayhteydet, keskellä tietokanta, alhaalla operatiiviset laitteet ja oikealla käyttöliittymään liittyvät moduulit. (11.)

3.2 Ignition Designer ja Perspective

Ignition Designer on työkalu, jonka avulla kehittäjä voi määritellä, hallita sekä rakentaa kehitysympäristöjä ja niiden käyttöliittymiä. Esimerkiksi Talotohtori-projektissa on neljä eri ympäristöä, joista yksi on kehittäjien testausympäristö, toinen kehittäjien kehitysympäristö, kolmas betaympäristö ja neljäs on tuotantoympäristö. Esimerkistä lisää kuvassa 3.



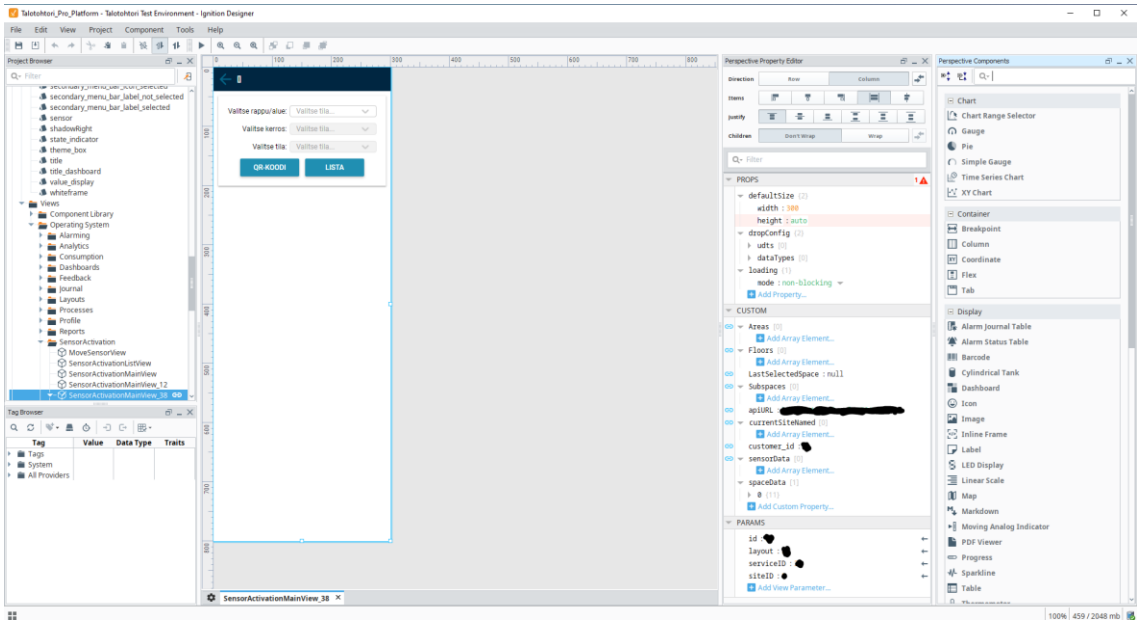
Kuva 3. Designer ja Talotohtorin kehitysympäristöt. Painamalla jotain näistä kehitysympäristöistä avautuu Ignition Perspective visualisointimoduuli, jossa on mahdollista rakentaa käyttöliittymää.

Ignition Perspective on yksi kahdesta Ignition Designerin visualisointimoduuleista. Perspective mahdollistaa käyttöliittymän rakentamisen. Tämä moduuli mahdollistaa responsiivisen käyttöliittymän rakentamisen eli Perspectiven kääntämiä sivustoja voi käyttää myös mobiililaitteilla. Perspectivessä on sisään rakennettuja elementtejä, joiden toiminnallisuudet toteutetaan joko python ohjelmointikielellä tai sitomalla haluttu tieto tiettyyn komponenttiin.

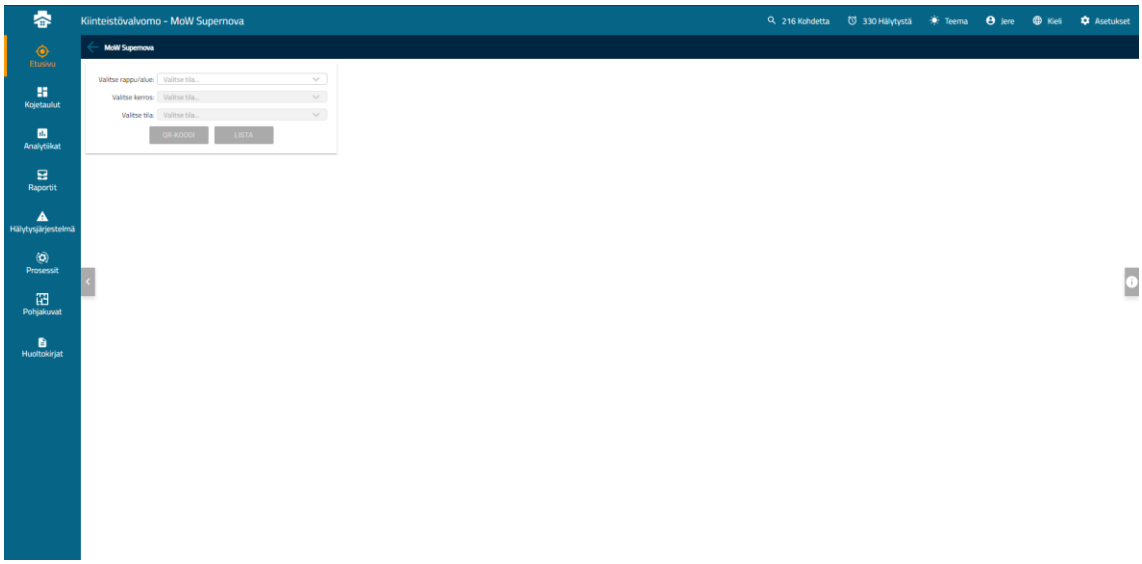
Perspectivessä käytetään asiakkaalle näkyvästä sivusta ilmaisua istunto. Istunto koostuu Ignitionin luomista valmiista sivu- ja näkymäelementeistä. Suurin elementti on istunto ja pienin on komponentti. Alkaen pienimmästä eli komponentista. Komponentteja ovat yksittäiset elementit, jotka esittävät jotain tietoa. Esimerkiksi painikkeet, syöttökentät,

taulukot ja muut elementit, joiden kanssa käyttäjät ovat vuorovaikutuksessa. Kaikki komponentit täytyy sijoittaa kontin sisään. Kontit määrittävät komponentin sijainnin, sekä kuinka komponentin tulisi reagoida sivun koon vaihteluun.

Seuraavana on näkymä. Näkymä sisältää kontteja. Näkymän sisälle voi lisätä myös toisen näkymän. Kun näkymä lisätään näkymän sisälle, halutaan yleensä luoda näkymäkokonaisuus. Näkymäkokonaisuus voi sisältää seuraavia elementtejä: sivupalkin, ponnahdusikkunan tai joukon pienempiä elementtejä. Tästä muodostuu sivu. Sivu on kokoelma näkymiä eli yksi välilehti selaimessa ja kaikki sen sisältö. Suurin elementti on istunto. Kokoelma avoimia sivuja yhdestä selaimesta. Perspective kääntää istunnon näkymät HTML5-sivustoiksi. (8.) Kuvassa 4 on esimerkki Perspectiven kehitysympäristön näkymästä ja sen jälkeen kuvataan (ks. kuva 5) kehitysympäristön näkymää vastaavasta käännetystä sivustosta.



Kuva 4. Ignition Perspective. Kuvassa vasemmalla projektipuu, keskellä komponenteista muodostettu näkymä ja oikealla muuttujaeditori sekä komponenttivalikoima.



Kuva 5. Ignition Perspectiven HTML5 käännetty versio yllä olevasta kuvasta (ks. kuva 4).

3.3 Robot Framework & SeleniumLibrary

Robot Framework on saanut alkunsa Nokia Networksin toimesta, se on julkaistu avoimesti käyttöön vuonna 2008. Robot Framework on avoimeen lähdekoodiin perustuva geneerinen automaatiokehys. Automaatiokehystä käytetään testiautomaatioon ja robottiprosessien automatisointiin. Robot Framework on erittäin joustava, sillä sen voi integroida lähes mihin tahansa ohjelmistotyökaluun ja siihen on rakennettu lukuisia kirjastoja sekä työkaluja, jotka tukevat sen käyttöä. Yksi esimerkki kirjastosta on SeleniumLibrary, jota tässäkin projektissa on käytetty ja se esitellään tässä osiossa. Kirjastojen ja työkalujen kehityksen toteuttaa kolmannet osapuolet. Lisäosat on kehitetty joko Javalla tai Pythonilla. Robot Framework on erittäin suosittu ohjelmistokehityksessä ja sitä käyttää monet alan johtavat yritykset, kuten Siili_ ja Tieto. Robot Framework on käyttöjärjestelmästä ja sovelluksesta riippumaton, eli se toimii kaikilla alustoilla.

Automaatiokehys on toteutettu ohjelmointikieli Pythonilla, se tukee ohjelmointikieliä Python 2:sta, Python 3:sta, Jythonia, IronPythonia sekä PyPyä. Robot Framework toimii siis Pythonilla. Robot Framework vaatii Pythonin sekä PIP:n asennuksen ennen Frameworkin asennusta. Python on funktionaalinen ja olio -ohjelmointikieli, joka on samankaltainen kuin esimerkiksi Java (12). PIP on Pythonin lisäosien asennuksiin tarkoitettu työkalu (13).

Robot Frameworkin syntaksi on luotu helposti lähestyttäväksi, se hyödyntää ihmisille luettavia avainsanoja. Esimerkkikoodissa 1 on testisarja, joka testaa kriteereiden mukaisen kirjautumisen. Testin syntaksi perustuu avainsanoihin. Avainsanat ovat monikäyttöisiä, eli niitä voidaan käyttää toisten avainsanojen sisällä. Tämän avulla testauksen tarkat yksityiskohdat voidaan naamioida selkokieliseksi. Esimerkiksi testin vaihe *Then user is logged in* ei näytä kovin tarkkaa selitystä, mitä testin sisällä todella tapahtuu, mutta sen yleiskuva tulee lukijalle selväksi. (14.) Testiin voi liittää erilaisia tunnisteita. Esimerkissä [Tags] rivillä on erilaisia tunnisteita, joiden avulla voidaan ajaa kaikki testit, joilla on tietty tunniste. Esimerkiksi kaikki testit, joilla on tunniste Dev, voidaan ajaa samaan aikaan.

```
*** Settings ***
Documentation       Suite description
Library             SeleniumLibrary
Resource            ../common/CommonKeys.robot
Suite Teardown      Close All Browsers

*** Test Cases ***
Login - With proper credentials
    [Tags]  Nightly  Login  Dev
    Given I Open browser to "talotohtori"
    When I click "Initial sign in"
        and I input username "robot@test.fi"
        and I click "continue"
        and I input password "*****”
        and I click "continue"
    Then user is logged in
    [Teardown]  Close All Browsers
```

Esimerkkikoodi 1. Robot Framework testisarja, joka testaa Talotohtori-palvelun kirjautumista. Testisarja käyttää avainsanoja tiedostosta CommonKeys.robot.

Avainsanat voi esitellä erillisessä tiedostossa, ne voi tuoda testisarjaan erillisenä resurssina. Tämä auttaa pitämään testisarjat siisteinä ja mahdollistaa koko projektin avainsanojen löytymisen yhdestä paikasta. Alla on esimerkki resurssitiedostosta, jossa näkyy *Then user is logged in* sisältö. Tästä on havaittavissa avainsanoilla naamioimisen hyödyt. Syntaksi testisarjassa on siistimpää ja resurssitiedoston avulla voidaan jättää epäkäytännölliset muuttujat ja avainsanat resurssitiedostoon.

```
*** Variables ***
${TT_LOGO_HOMEBUTTON}    //*[@data-component-path="T[0]$0:0:0.0:0:0"]

*** Keywords ***
user is logged in
    Wait until element is visible    ${TT_LOGO_HOMEBUTTON}    timeout=15
    Page Should Contain Element      ${TT_LOGO_HOMEBUTTON}
```

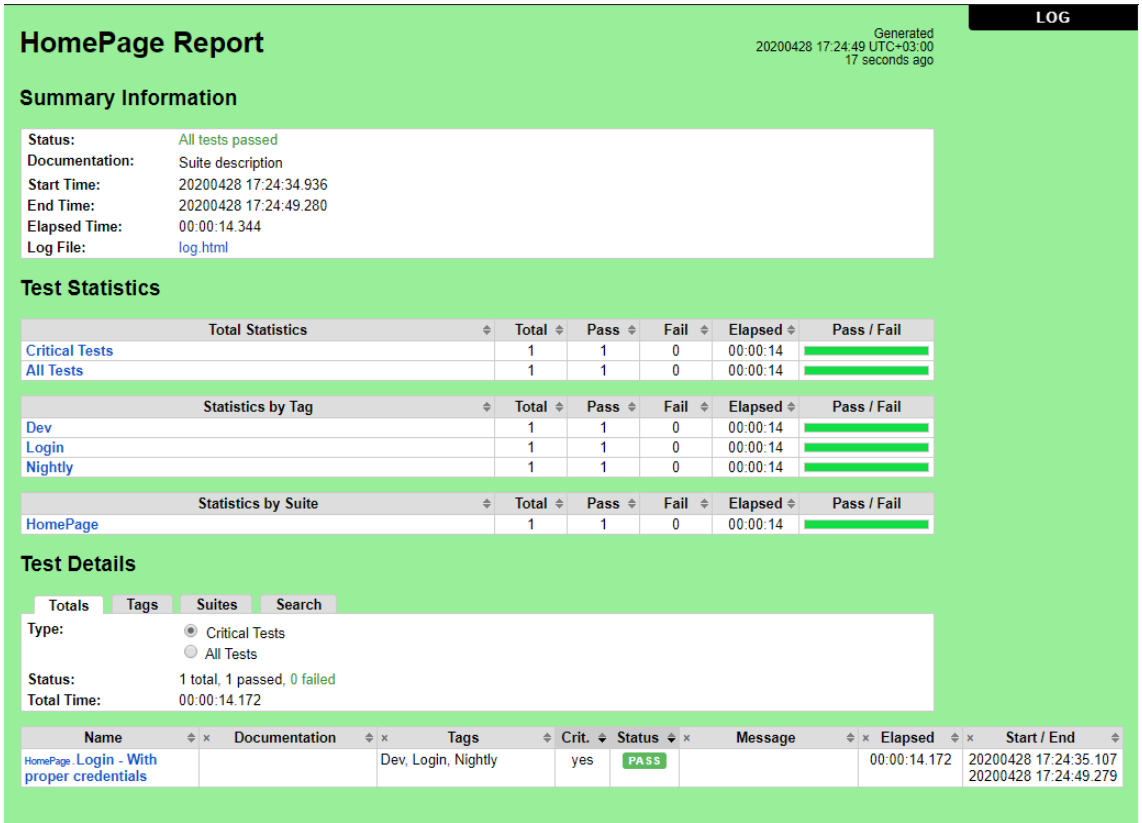
Esimerkkikoodi 2. Robot Frameworkin resurssitiedosto, jossa esitellään avainsanoja ja niiden muuttujia. user is logged in käyttää SeleniumLibraryn tuomia avainsanoja.

SeleniumLibrary on Selenium WebDriveriin pohjalta rakennettu Robot Framework kirjasto web-testaamiseen. Selenium WebDriver mahdollistaa selaimen käytön automatisoinnin (15). SeleniumLibrary sisältää valmiita avainsanoja, joilla voi olla vuorovaikutuksessa verkkosivun tietyn elementin kanssa. Avainsanat käyttävät argumenttina paikanninta, joka määrittää, miten elementti löydetään verkkosivulta ja mitä sillä tehdään (16). Alla on koodiesimerkki elementin paikantamisesta ja sen painamisesta.

```
click element                  //span[text ()='CONTINUE TO SIGN IN']
```

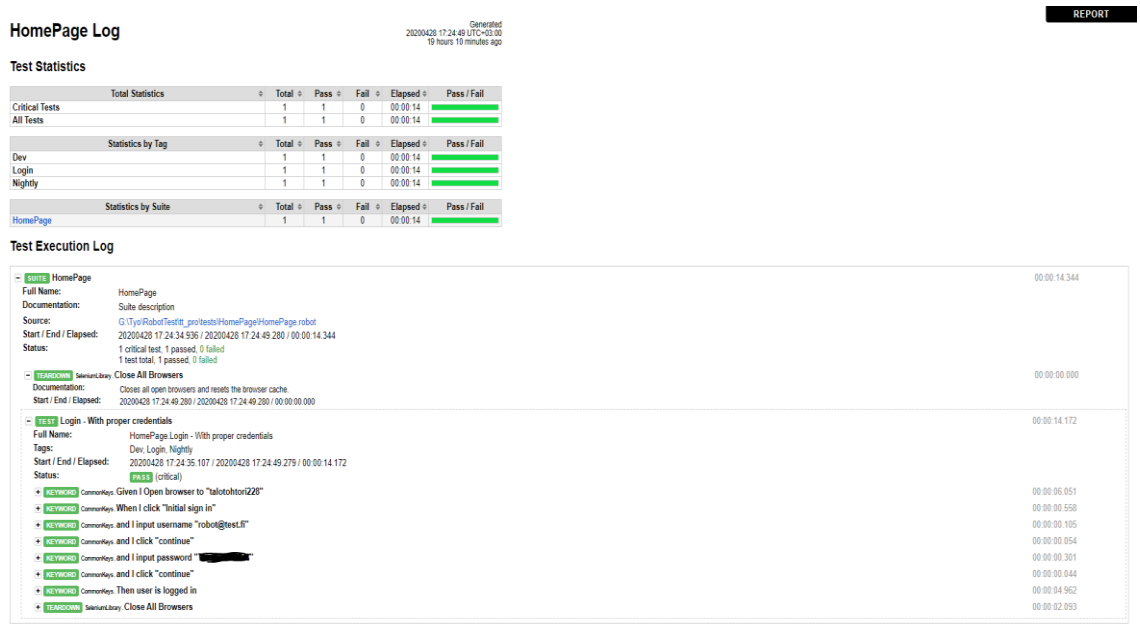
Esimerkkikoodi 3. SeleniumLibrary syntaksi, jossa esitellään avainsana click element. Tässä paikannetaan verkkosivulta elementti Span, joka sisältää tekstin 'CONTINUE TO SIGN IN' ja painetaan siitä.

Robot Framework tuottaa erittäin kattavan palautteen testien suorittamisen tapahtumista. Automaatiokehys tuottaa kaksiosaisen palautteen, joista ensimmäinen on raportti. Raportti esittää tilastoja muun muassa yhteen testiin kuluneesta ajasta, testien läpäisy- ja epäonnistumissuhteet. Kuvassa 6 on 1. esimerkkikoodin suorituksen luoma raportti.



Kuva 6. Suoritusraportti, jossa kerrotaan testien onnistumiset, epäonnistumiset ja niiden suhteet. Raportista on nähtävissä myös kokonaissuoritusaika sekä jokaisen testitapauksen yksittäiset suoritusajat.

Toinen automaatiokehyksen tuottama palaute on lokitiedosto. Lokitiedostossa on tarkat tiedot testin avainsanojen suorittamisen jokaisesta vaiheesta. Sen avulla on helppo löytää vikatilanteet tai tarkastaa mahdolliset suorituksen aikana otetut näyttökuvat. Kuvassa 7 on 1. esimerkkikoodin suorituksen muodostama lokitiedosto.



Kuva 7. Lokitiedosto, jossa on eritelty testin jokaisen avainsanan jokainen vaihe. Lokitiedostosta on löydettävissä myös jokaisen vaiheen kesto.

3.4 PyCharm

Tässä työssä Robot Framework -testien kirjoittamiseen on käytetty monialustaista PyCharm-kehitysympäristöä. PyCharm on julkaistu vuonna 2010 tšekkiläisen yrityksen JetBrainsin toimesta, se on suunnattu Python ohjelmointikielelle. PyCharm on integroitu kehitysympäristö. PyCharm tarjoaa muun muassa syntaksin ja virheiden korostamisen sekä integroidun versionhallinnan Git:n kanssa. Lisäksi siihen voi ladata lukuisia lisäosia esimerkiksi järjestämään tekstin muotoa. (17.)

3.5 GIT

Git on ilmainen ja avoimeen lähdekoodiin perustuva hajautettu versionhallintajärjestelmä. Versionhallinta on suunniteltu pitämään kirjaa tehdyistä muutoksista ja tallentamaan niiden vanhemmat versiot lähdekoodista. (18.) Git:n ideana on projektin samanaikaisen kehityksen mahdollistaminen monella eri päätteellä. Git mahdollistaa useiden paikallisten haarojen luomisen, ne voivat olla täysin riippumattomia toisistaan. Haaroja voi luoda, yhdistää ja poistaa. Tämä mahdollistaa turvallisen ohjelmistokehityksen suuressakin projektissa. (19.) Git on kilpailijoihinsa nähden nopea. Hajautetun järjestelmän nopeus perustuu lähes kaikkien toimintojen suorittamiseen paikallisesti. Keskitetyt järjestelmät joutuvat kommunikoimaan jatkuvasti tietyn palvelimen kanssa. (20.)

3.6 Jira-ohjelmisto

Jira on Atlassianin patentoitu työn hallitsemiseen suunniteltu työkalu. Alun perin Jira oli suunniteltu vikojen ja ongelmien seuraamiseen, mutta nykyään se on kehittynyt tehokkaaksi työnhallinnan työkaluksi monipuolisille käyttötapauksille. Jiraa voi hyödyntää esimerkiksi vaatimusten ja testitapausten hallintaan tai vaikka ketterään ohjelmistokehitykseen. (21.)

Enermix Oy on hyödyntänyt Jiraa muun muassa ketterässä ohjelmistokehityksessä, siihen Jira tarjoaa ketterien menetelmien kanban- sekä scrumtaulut. Ketterässä ohjelmistokehityksessä pyritään työskentelemään lähellä asiakasta. Lyhyissä yhdestä neljään viikkoon kestävissä iteraatioissa, joista jokaisen päätteeksi julkaistaan toimiva ohjelmistoversio. (22.) Taulut toimivat asioidenhallintapaneeleina, joissa asiat kartoitetaan muokattavissa oleviin työnkulkuihin. Työnkulku voi olla esimerkiksi määrittely, kehityksessä, kehitys valmis, testaus ja valmis. Taulut tuovat läpinäkyvyyttä projektin etenemiseen, se näyttää jokaisen asian tilan.

Asiat ovat yksittäisiä töitä, jotka on suoritettava. Asia voi olla esimerkiksi suuri ominaisuus eli eepos, käyttäjätarina tai yksittäinen testi. Asiat sisältävät yksityiskohdat asiasta, aikataulun ja muistutukset. Asioihin voi lisätä aliasioita, joten yksi asia voidaan pilkkoa pienemmiksi osiksi koko tiimin kehitettäväksi. Asialle määritellään aina raportoija sekä

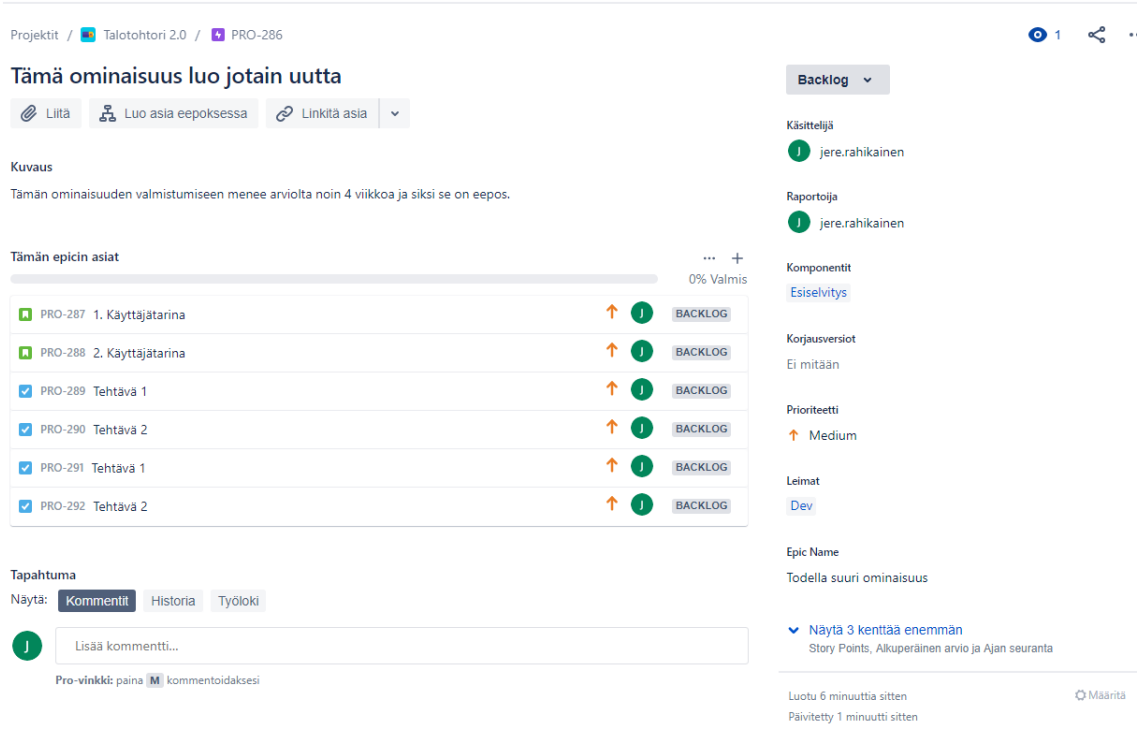
käsittelijä. Jira tarjoaa myös kattavat ja reaaliaikaiset suoritusraportit tuottavuuden mittaamiseen. Raportteja ovat muun muassa sprinttiraportit ja käytettävänä olleen ajan vertaus käytettyyn aikaan.

Tämän työn kannalta tärkeitä ominaisuuksia ovat Jiran lisäosa Xray ja asian monipuolisuus. Xraystä kerrotaan tässä työssä myöhemmin lisää. Asiaa voi käyttää monella tapaa. Uutta ominaisuutta lisättäessä on tärkeää määrittää, minkä kokoisesta asiasta on kyse. Jira tarjoaa alla olevassa taulukossa näkyvät vaihtoehdot asiatyyppien määrittämiseen suurimmasta pienimpään. Asioiden määrittely suurimmasta pienimpään auttaa luomaan projektiin selkeämpää hierarkiaa ja sen avulla ison ominaisuuden valmistumista on helpompi seurata ja osa ominaisuudesta voidaan jo julkaista. Näin jokaisen julkaisun yhteydessä asiakas saa aina uutta. (23; 24.)

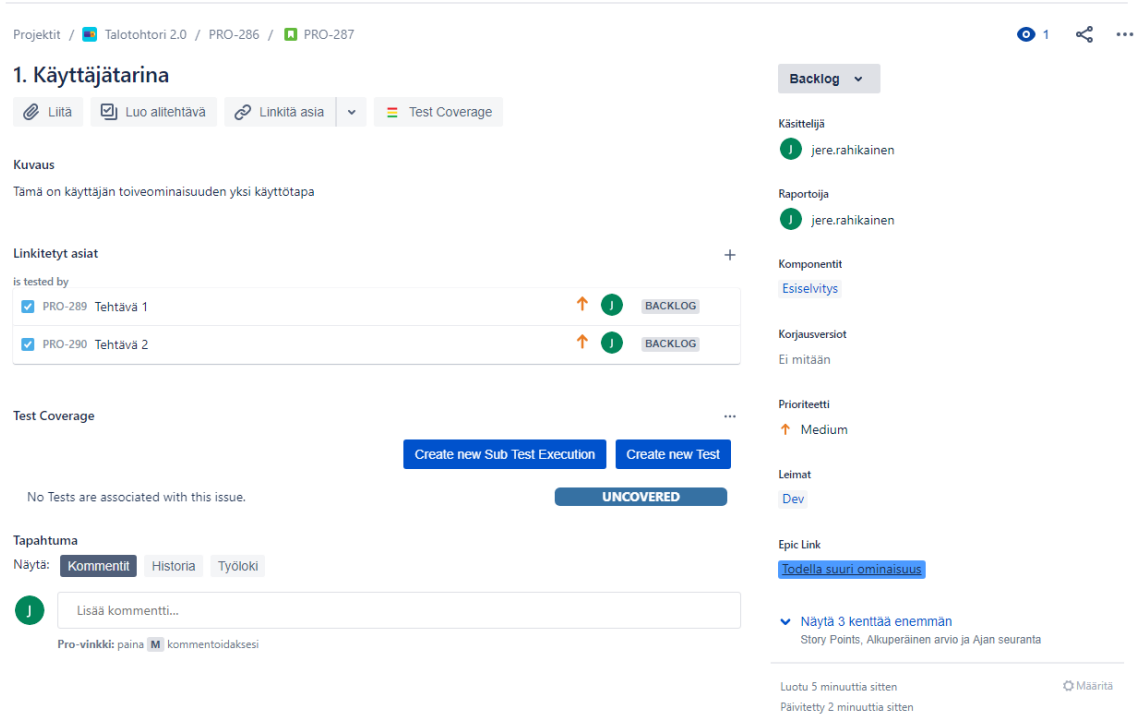
Taulukko 1. Jiran asiatyypit ja määritelmät suurimmasta pienimpään.

Asiatyyppi	**Määritelmä**
Eepos	Suurin kokonaisuus, joka voidaan jakaa pienempiin tarinoihin tai käyttäjätarinoihin
Tarina/Käyttäjätarina	Vaatimus tai tarve käyttäjän näkökulmasta, joka voidaan jakaa pienempiin tehtäviin
Tehtävä	Yksittäinen tehtävä tarinan sisällä tai esimerkiksi yksittäisen tehtävän sisäinen TO-DO

Kuvissa 8 ja 9 on käytännön esimerkki Jirassa muodostetusta eepoksesta sekä käyttäjätarinasta.



Kuva 8. Jiran eepos, johon on liitetty käyttäjätarinoita sekä käyttäjätarinoiden tehtäviä.



Kuva 9. Jiran käyttäjätarina, johon on liitetty kaksi tehtävää. Oikealla kuvassa kohdalla Epic Link näkyy myös tieto, mihin eepokseen käyttäjätarina on liitetty.

3.7 Xray

Xray on Jiraan liitettävä testienhallintatyökalu, joka mahdollistaa testauksen suunnittelun, testien suunnittelun, testien toteutuksen ja testiraportit Jirassa asioina. Tämä testienhallintatyökalu mahdollistaa testauksen tilanteen seuraamisen reaaliaikaisesti. Testauksen tilanteista on mahdollista luoda raportteja sekä kattavuusanalyyseja. (25.) Xrayn on kehittänyt vuonna 2003 perustettu Xpand IT, jolla on neljä toimipisteittä ympäri maailmaa.

Viitaten tässä dokumentissa aiemmin esiteltyyn taulukkoon 1. Xray tuo Jiran asiatyyppeihin viisi uutta tyyppiä. Näitä asiatyyppejä voi kontrolloida samalla tavalla kuin muitakin asiatyyppejä. Taulukossa 1 on esitelty uudet asiatyypit.

Taulukko 2. Xrayn Jiraan tuomat asiatyypit ja niiden määritelmät. (26.)

Asiatyyppi	**Määritelmä**
Test Plan	Kertoo mitkä testi on suunniteltu ajettavaksi kussakin ohjelmistoversiossa.
Test Set	Joukko testejä. Esimerkiksi kaikki testit, jotka liittyvät johonkin tiettyyn ominaisuuteen.
Precondition	Kertoo mitä täytyy tehdä ennen, kun testin vaiheet voi suorittaa
Test	Testi. Testi sisältää testin vaiheet, toiminnot ja odotetut tulokset.
Test execution	Yksittäisen testin ajon palaute ja tulos. Näitä voi tuoda Xrayn rajapinnan kautta suoraan esimerkiksi Robot Frameworkista

Xray tukee manuaalista ja automaatiotestausta. Automaatiotesteistä Xray tukee täysin Cucumber-testejä sen alkuperäisellä syntaksilla. Tämä tarkoittaa sitä, että Cucumber testejä voi luoda ja ajaa suoraan Jirassa. Cucumber on testaustyökalu, joka tukee käyttäytymiseen perustuvaa kehitystä. Käyttäytymiseen perustuva kehitys kannustaa koko yrityksen henkilöstöä yhteistyöhön ohjelmistoprojektissa. Cucumberin syntaksi on luotu niin helpoksi, kuka tahansa voi ymmärtää sitä.

Xray tarjoaa oman rajapinnan, jonka avulla on mahdollista yhdistää jatkuvan integraation työkaluja Jiraan. Rajapinta mahdollistaa myös automaatiotestien ja niiden palautteiden tuomisen automaatiokehyksistä suoraan Jiraan. Esimerkiksi Robot Frameworkin tuottama palaute voidaan tuoda Jiraan näkyville rajapinnan avulla. (25.)

3.8 Testausmenetelmiä

3.8.1 Manuaalinen testaus

Manuaalisella testauksella tarkoitetaan ohjelmiston testausta, jossa testaaja suorittaa testejä itse. Tällä pyritään löytämään mahdolliset virheet ohjelmistosta. Testaaja tarkistaa sovelluksen kaikki olennaiset ominaisuudet. Testaaja suorittaa ja tuottaa kaikki testitapaukset sekä testiraportit manuaalisesti ilman minkään automaatio-ohjelmiston apua. (27.)

3.8.2 Automattinen testaus

Automaattisella testauksella tarkoitetaan ohjelmiston testausta, jossa testaaja ohjelmoi testin suorituksen automaattiseksi. Testaaja hyödyntää testaukseen luotuja automaatiotyökaluja testien ohjelmoinnissa. Automatisoinnilla pyritään vähentämään testaukseen käytettyä aikaa ja tehostamaan virheiden löytämistä. Automaattinen testaus perustuu esiohjelmoituun testiin, jossa automaattisen testin tulosta verrataan oletettuun tulokseen. Vertailun tuloksena testaaja tietää, toimiiko sovellus odotetusti vai ei. Automaattinen testaus ei kuitenkaan poista manuaalisen testauksen tarvetta täysin. Manuaalista testausta tarvitaan esimerkiksi testien ohjelmoinnissa. (27.)

Automaatiotestaamiseen on monia tapoja, joilla voi minimoida ohjelmiston virheiden määrää. Seuraavaksi käydään lyhyesti läpi erilaisia vaihtoehtoja testaamiselle ja mitä eroja niillä on.

Yksikkötestaus

Yksikkötestit ovat tarkoitettu testaamaan ohjelmiston lähdekoodia, esimerkiksi luokkia, komponentteja tai koodin tiettyä toiminnallisuutta. Yksikkötestit tarkistavat, toimiiko koodi suunnitellulla tavalla. Yksikkötestit kirjoitetaan yleensä ohjelmoijan toimesta käsin. (28; 29.)

Integraatiotestit

Integraatiotestit tarkastelevat ohjelmiston ja ohjelmistosta irtonaisten moduulien tai palvelujen toimivuutta yhdessä. Integraatiotesti voi testata esimerkiksi käyttöliittymän ja tietokannan toimivuutta yhdessä. (28.)

Funktionaaliset testit

Funktionaaliset testit keskittyvät ohjelmiston toiminnallisuuden testaamiseen. Testit testaavat siis toiminnallisuuden tuottaman tuloksen. Funktionaaliset testit eroavat integraatiotesteistä sillä, että integraatiotesti testaa toimiiko yhteys tietokantaan, kun taas funktionaalinen testi testaa, mikä on tietokannan vastauksen arvo. (28.)

Käyttöliittymätestaus

Käyttöliittymätestaus yrittää matkia loppukäyttäjän käyttäytymistä käyttöliittymässä. Yleensä käyttöliittymän testitapaukselle määritellään jokin polku, jota pitkin testin pitäisi kulkea. Polku voi olla esimerkiksi sovellukseen kirjautuminen. (28.)

Suorituskykytestaus

Suorituskykytesteillä testataan ohjelmiston suoriutumista erilaisista kuormituksista. Suorituskykytestillä voi testata esimerkiksi vasteaikoja, kun ohjelmisto kohtaa suuren määrän käyttäjiä tai pyyntöjä. (28.)

Savutestaus

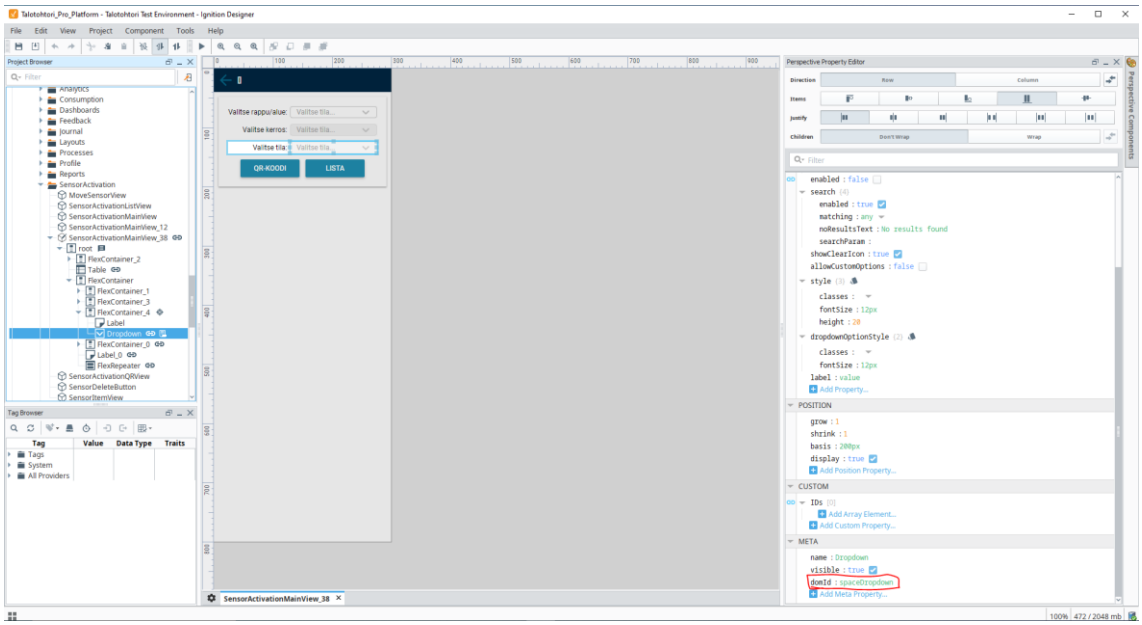
Savutestaus on tarkoitettu testaamaan ohjelmiston tärkeimpiä ominaisuuksia. Savutestauksen avulla voi esimerkiksi selvittää, toimiiko kaikki sovelluksen tärkeimmät ominaisuudet vielä uuden version päivityksen jälkeen. (28.)

4 Työn kulku

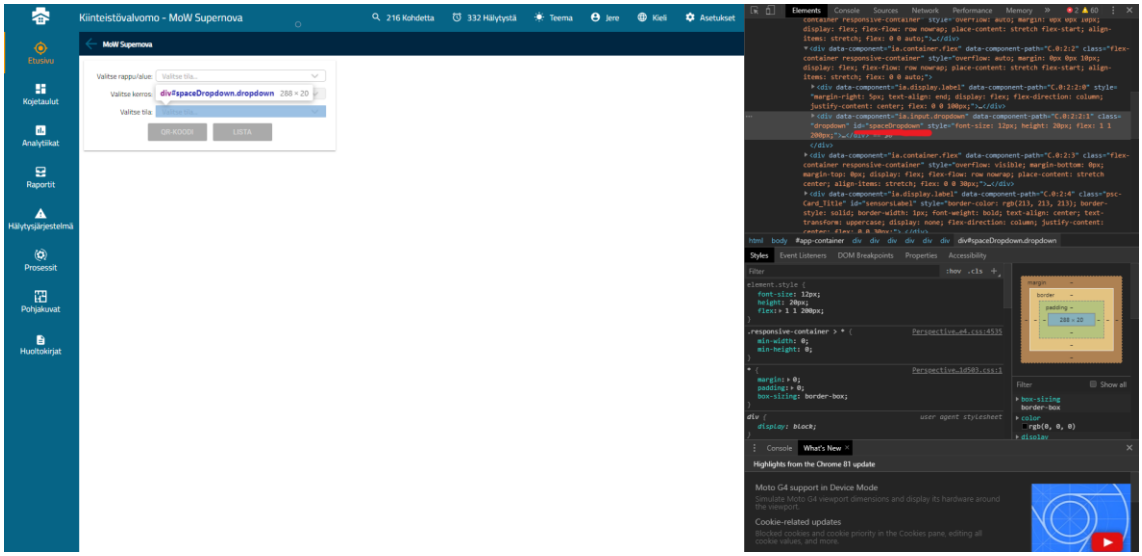
Työn alussa pohdittiin eri testausteknologioiden sopivuutta Ignition Perspectiveen. Yksi haasteista Perspectiven kanssa oli HTML5-käännös. Käännöksen sisään ei ollut keinoa syöttää elementin tunnistamiseen tarkoitettua tunnistetta. Käännöksen aikana elementtien sijainti saattaa muuttua sisällön perusteella, joten tietyn tiedon tunnistaminen sivulta voi olla testausautomaatiolla mahdotonta. Toinen haaste Perspectiven kanssa oli yksikkötestauksen harjoittaminen. Yksikkötestausta pystyy tällä hetkellä harjoittamaan ainoastaan manuaalisesti, tämä on täydellisesti ohjelmoijan käsissä. Manuaalisessa testauksessa ei voi välttyä virheiltä, se näkyy ohjelmiston kehityksessä.

Ensimmäiseen haasteeseen löydettiin Ignitionin keskustelupalstalta ratkaisu. Elementteihin pystyi syöttämään sittenkin tunnisteen. Perspectivessä komponenteilla on oma

META-kenttä, johon täytyi luoda uusi muuttuja nimellä domId ja sen arvoksi sitten haluama tunniste. (30.) Tämän avulla elementit saivat oman tunnisteen HTML5-käännökseen. Alla havainnollistavat kuvat domId:n toiminnasta (ks. kuva 10, kuva 11).



Kuva 10. Ignition Perspectiven meta kenttä. Oikealla alhaalla META-kentän ja domId:n esittely.



Kuva 11. Talotohtorin käyttöliittymä ja Chrome selaimen kehittäjän konsoli. Kuvassa näkyy vasemmalla elementti, johon kuvassa 10 syötettiin domId. Sama domId löytyy kehittäjän konsolista kyseisen elementin tunnisteesta. Tästä voi todeta, että tämän elementin voi tunnistaa kyseisellä domId:llä.

Toiseen haasteeseen ei tämän työn aikana löydetty parempaa ratkaisua, joten tässä työssä keskitytään käyttöliittymätestaamiseen. Käyttöliittymätestaamisella on mahdollista löytää joitain toiminnallisuuden virheitä. Esimerkiksi on mahdollista muodostaa testi, jossa testataan, poistuuko poistettu tieto määritetyltä listalta.

Kuten lähtökohdissa oli todettu, työ sai suuntansa yhteisessä tapaamisessa Etteplanilla. Heillä oli kokemusta Robot Frameworkin luotettavuudesta käyttöliittymätestaamisessa ja suosittelivat vahvasti sitä. Vahvojen suosituksien takia valittiin käyttöliittymätestaamiseen Robot Frameworkin SeleniumLibrary lisäosalla. SeleniumLibrary tuo lukuisia valmiita avainsanoja, joille löytyy hyvät ja selkeät dokumentaatiot. Robot Framework on Python pohjainen, joten kehitystyökaluksi valitsimme PyCharmin, jossa oli hyvät ominaisuudet muun muassa syntaksivirheiden korjaamiseen sekä versiohallintatyökalu Git:iin.

Automaatiotestausta varten perustettiin oma testausympäristö. Uusi ympäristö sisälsi käytännössä kloonin Talotohtorin kehitysympäristöstä. Suunnitelmana oli ensin luoda manuaalinen työnkulku automaatiotestaukselle ja tämän jälkeen, kun testikattavuus nousee, siirretään järjestelmä automaattiseen työnkulkuun. Manuaalinen työnkulku tarkoittaa sitä, että joukko automaatiotestejä käynnistetään ihmisen toimesta tiettyyn aikaan. Testien tulokset lähetetään ihmisen toimesta työnseurantaan. Automaattisella työnkululla tarkoitetaan, että joukko automaatiotestejä käynnistetään ajastetusti tiettyyn aikaan ja testien tulokset lähetetään automaattisesti työnseurantaan.

Seuraavaksi esitellään automaatiokehyksen paikallisen ajoympäristön pystytys eli Pythonin, Robot Frameworkin, SeleniumLibraryn, Git:in ja PyCharmin asennus. Tämän jälkeen esitellään työvaiheen määrittely ja suunnittelu sekä toteutus sisältäen Jiran ja Xrayn roolit.

Asennus on aloitettava Pythonin asennuksesta, sillä se on koko automaatiokehyksen pohja. Pythonin lataussivustolta on saatavilla asennustiedostoja Windowsille, Macille sekä Linuxille. Työssä käytetään Windowsia, joten ladattiin sille sopiva asennustiedosto ja asennettiin Pythonin uusin versio 3.8.2 ohjelman sisältämien ohjeiden mukaan. (31.) Seuraavaksi asennettiin Git ja PyCharm, jonka asennukset toimivat samankaltaisesti, kuin Pythonin. Git:n asennustiedoston voi ladata Git:n lataussivulta (32). PyCharmin

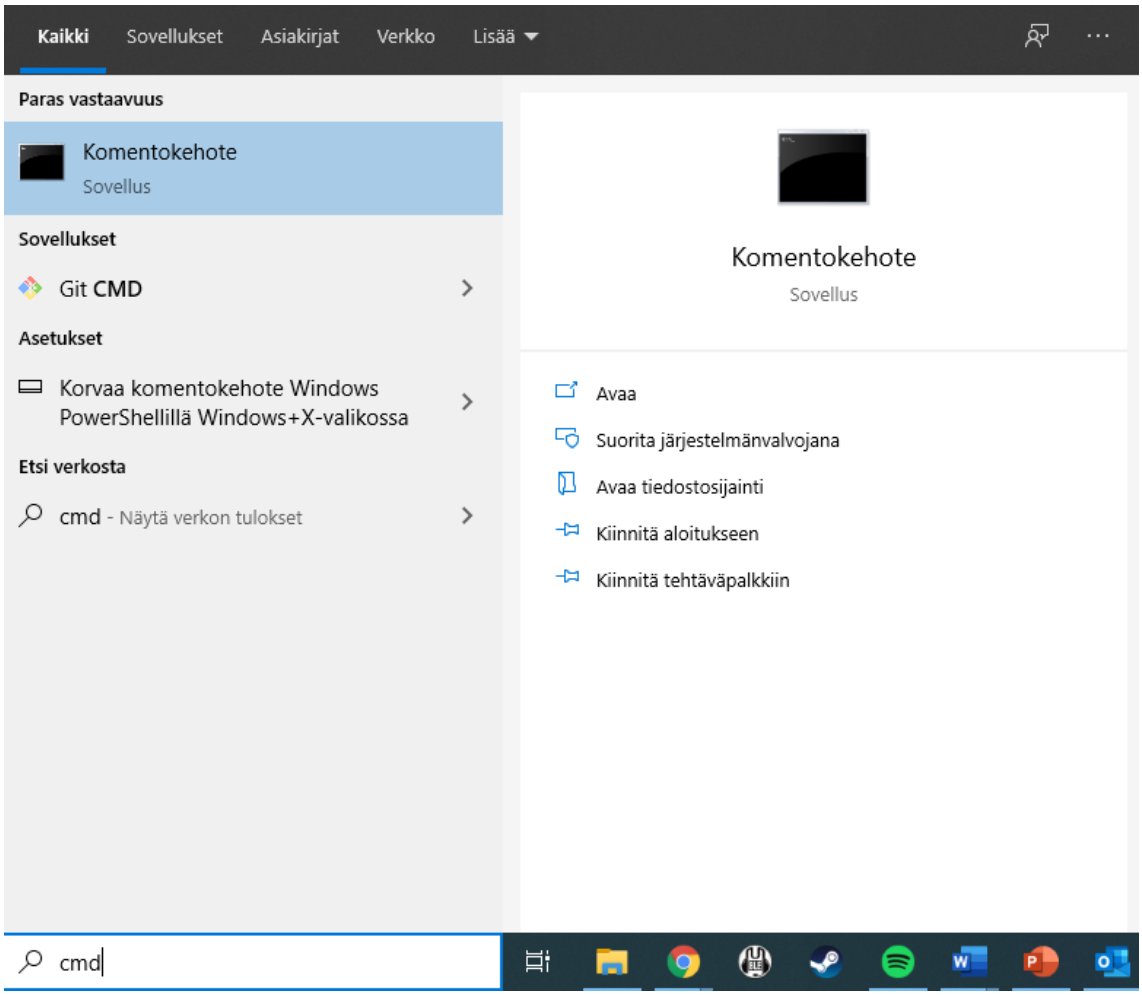
asennustiedoston voi ladata JetBrainsin lataussivulta (33). Pythonin asennuksen jälkeen on syytä määritellä Python ja sen lisäosat Windowsin ympäristömuuttujiin. Ympäristömuuttujia pääsee muokkaamaan kirjoittamalla Windowsin hakukenttään Muokkaa järjestelmän ympäristömuuttujia. Ympäristömuuttujiin kannattaa lisätä Python.exe sekä Pythonin asennuskansiosta löytyvä scripts kansio. Näiden avulla Pythonin ja Robot Frameworkin sisäiset komennot toimivat komentokehotteessa.

Robot Framework asennetaan Pythonin komentokehotteessa komennolla:

```
pip install robotframework
```

Esimerkkikoodi 4. Robot Frameworkin asennuskomento. (34.)

Komentokehotteeseen pääsee avaamalla Windowsin hakukentän, kirjoittamalla sinne cmd ja painamalla ENTER. Alla olevassa kuvassa on esimerkki tästä.



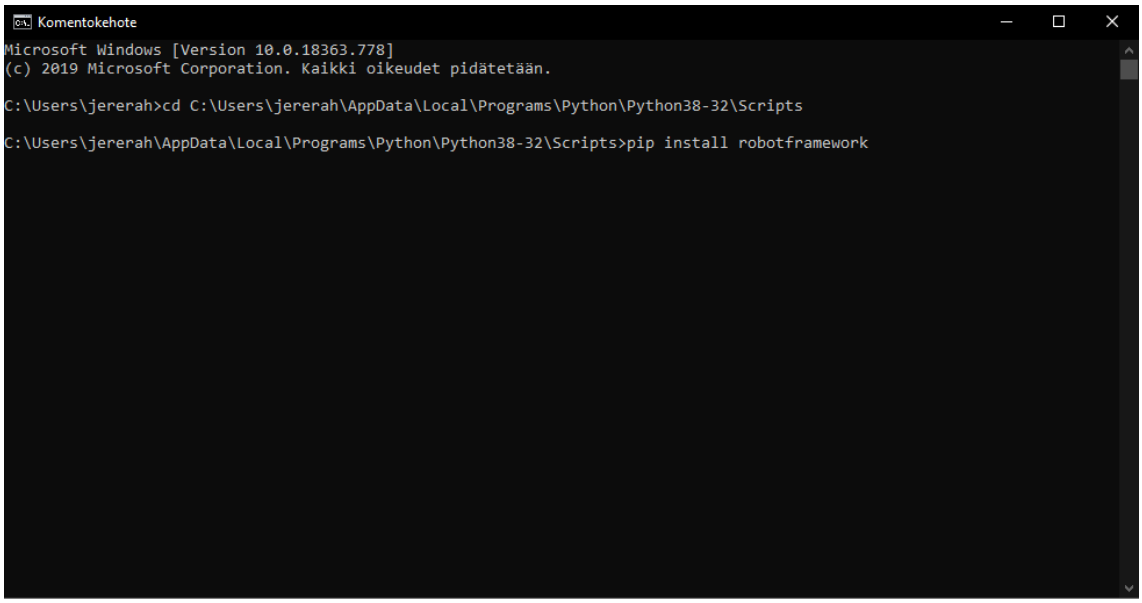
Kuva 12. Kuvassa esitellään, kuinka löydetään komentokehote.

Seuraavaksi komentokehotteessa navigoidaan Pythonin lisäosien asennuskansioon. Navigointi tapahtuu komennolla *cd <tiedostopolku>*. Jos Python on asennettu vakiotiedostopolkuun, löytyy se alla olevan esimerkkikoodin polusta.

```
cd C:\Users\<WindowsKäyttäjätunnus>\AppData\Local\Programs\Python\Python38-32\Scripts
```

Esimerkkikoodi 5. Pythonin lisäosien asennuspolku ja navigointikomento cd.

Navigoinnin jälkeen komentokehotteeseen kirjoitetaan 4. esimerkkikoodin sisältö ja näin Robot Framework on asennettu. Kuvassa 13 vielä esimerkki navigoinnista ja asennuksen komennosta.



Kuva 13. Windowsin komentokehotteessa navigointi ja Robot Frameworkin asennuskomento.

Robot Frameworkin asennuksen jälkeen on helppo asentaa samassa komentokehotenäkymässä Robot Frameworkin lisäosa SeleniumLibrary. SeleniumLibrary asennettaan 5. esimerkkikoodin komennolla.

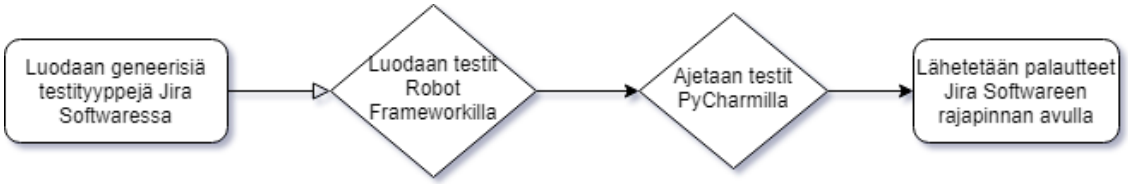
```
pip install --upgrade robotframework-selenium2library
```

Esimerkkikoodi 6. SeleniumLibraryn asennuskomento. (35.)

Automaatiotestausympäristön pystytyksen jälkeen opiskeltiin testien kirjoittamista. Opiskelussa käytettiin hyväksi Etteplanin henkilökuntaa ja Robot Frameworkista löytyviä oppaita. Opiskeluvaiheen jälkeen todettiin, että olisi tarpeellista tuottaa yhteiset ohjeistukset testaukseen ja siihen liittyviin asioihin. Ohjeistuksessa käsiteltiin seuraavia osa-alueita:

- Git:n tiedostopuun järjestys ja tiedostojen nimeämisen
- Robot Frameworkin resurssitiedoston
- testien nimeämisen
- testien tunnisteiden eli tagien nimeämisen ohjeistus.

Todettiin myös, että testien kattavuuden seuraamiseen olisi hyvä olla jokin työkalu. Työnhallinnassa käytettiin Jiraa ja siihen löydettiin lisäosa Xray, joka on tarkoitettu testienhallintaan. Xray tarjoaa hyvän dokumentoinnin ja sitä tutkiessa nousi esille, että Robot Frameworkin testien tulokset on mahdollista yhdistää Xrayn rajapinnan avulla Jiran työnseurantaan. Xray täydensi suunniteltua manuaalista työputkea lisäämällä tuloksien yhdistämisen työnseurantaan. Kuvassa 14 on luonnos manuaalisesti työputkesta.



Kuva 14. Automaatiotestauksen suunniteltu manuaalinen työputki. (36.)

Xrayn integraatio Jiraan toteutettiin noudattamalla automaattisen asennuksen ohjeita, jotka löytyivät Xrayn dokumentoinnista. (37.) Xrayn asiatyyppien käyttöönottoon löytyi ohjeistus myös Xrayn dokumentoinnista. Samassa ohjeessa kerrotaan, kuinka alkuperäiset asiatyypit on mahdollista valita kattavuusseurantaan. (38.)

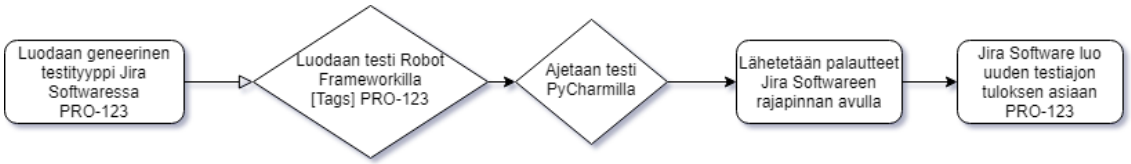
Xrayn ja Robot Frameworkin integraatio toteutettiin Xrayn rajapinnan avulla. Rajapintaan muodostetaan yhteys tunnistautumalla asiakkaan tunnisteella sekä asiakkaan salausavaimella, jotka on mahdollista luoda jokaiselle käyttäjälle erikseen Xrayn globaaleissa asetuksissa kohdassa API Keys. Tähän löytyy ohjeistus Xrayn dokumenteista. (39; 40.) Kun rajapintaan lähetetään tunnistus, se palauttaa tokenin, eli avaimen, jonka avulla avataan yhteys muihin rajapinnan toimintoihin. Avainta tarvitaan, kun lähetetään automaatiotestin palautetta eteenpäin PyCharmista. Rajapinta vaatii tulosten lähettämisen yhteydessä myös osoitetietoihin liitetyn projektiavaimen. Yhteyden muodostamiseen ja palautteen lähettämiseen tarvitaan hieman Python-koodia. Seuraavassa on koodiesimerkki tunnistautumiseen ja palautteen lähettämiseen. Koodissa on käytetty Pythonin lisäosaa requestia, joka mahdollistaa pyyntöjen lähettämisen rajapintoihin suoraan koodista.

```
import requests
// OSA 1
header = {
        'Content-Type': 'application/json',
    }
```

```
data = '{ "client_id": "**********",' \
           '"client_secret": "**********************************" }'

token = requests.post('https://xray.cloud.xpand-it.com/api/v1/authenticate',
headers=header, data=data)

// OSA 2
headers = {
        'content-type': 'application/xml',
        'Authorization': 'Bearer ' + token.json(),
    }

with open('output/output.xml','rb') as d:
    response = requests.post(
'https://xray.cloud.xpand-it.com/api/v1/import/execution/robot?project-
Key=PRO', headers=headers, data=d)
```

Esimerkkikoodi 7. Xray-rajapinnan tunnistautuminen ja testiajon tulosten lähettäminen. Osassa 1 tapahtuu tunnistautuminen ja avain tallennetaan token muuttujaan. Osassa 2 token liitetään osaksi asetuksia, output.xml-tiedosto osaksi dataosiota ja projektiavain liitetään osoitetietoihin.

Miten Jira sitten osaa liittää luodut testityypit Xrayn rajapinnan lähettämään output.xml-tiedostoon? Xray hyödyntää Robot Frameworkin testien tunnisteriviä löytämään oikeat asiat Jirasta. Asioiden tunnistamiseen Jira käyttää projektiavainta ja satunnaista lukua yhdistettynä. Sanotaan, että Jiran projektin tunnus on PRO. Yksi PRO-projektin sisäisistä asioista tunnistetaan tunnisteella PRO-123. Tätä tunnistetta hyödynnetään myös Xrayn rajapinnassa. Xray edellyttää, että kaikkiin Robot Frameworkin testeihin, jotka halutaan tuottavan tuloksen Jiraan, liitetään tunniste. Tunnisteen tulee olla sama, mikä sitä vastaava Jiran asian tunniste on. Kuvassa 15 on havainnointi tunnisteen merkityksestä.

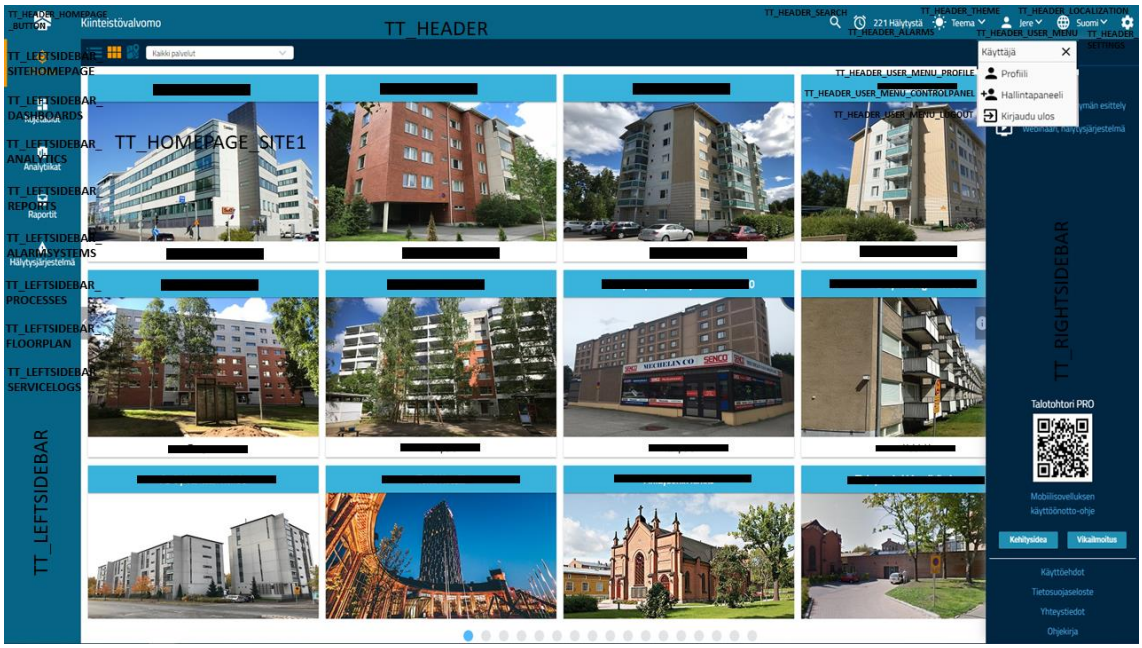


Kuva 15. Tunnisteen merkitys testityyppien ja automaatiotestien välillä.

Git:n tiedostopuun järjestyksellä pyrittiin kopioimaan Talotohtorin käyttöliittymän hierarkiaa. Ideana tässä oli, että testaajan olisi helppo löytää oikea tiedosto, johon uusi testi kirjotettaisiin. Tiedostojen nimeämisen ohjeistuksessa sovittiin, että kaikki erillissanat tiedoston nimessä alkaa isolla kirjaimella. Esimerkiksi TämäOnTiedosto.txt. Tämä lisättiin README.txt tiedostoon, joka löytyy projektin juuresta. Samaiseen tekstitiedostoon lisättiin myös ohjeistus Git:n yleisistä käytännöistä. Sovittiin, että jokainen testien kehittäjä

kehittää testit omassa haarassaan, josta ne kopioidaan päähaaraan. Muistutettiin myös, että aina ennen tallentamista on syytä ladata päähaara, ettei tallennuksissa tule sekaannuksia.

Robot Frameworkin resurssitiedoston järjestyksen ylläpitämiseksi sovittiin elementtien tunnisteiden nimeämisten yhtenäistämisestä. Kuvassa 16 on esimerkkikuva, jonka perusteella pyritään yhdistämään elementtien tunnisteiden nimeäminen. Sovittiin myös, että avainsanojen muuttujat nimetään yhtenäisesti. Esimerkiksi ${username}, eli muuttujan nimessä kaikki kirjaimet pienellä. Ohjeistukset kirjattiin samaan README.txt-tiedostoon kuin Git:n ohjeistukset.



Kuva 16. Elementtien tunnisteiden nimeäminen.

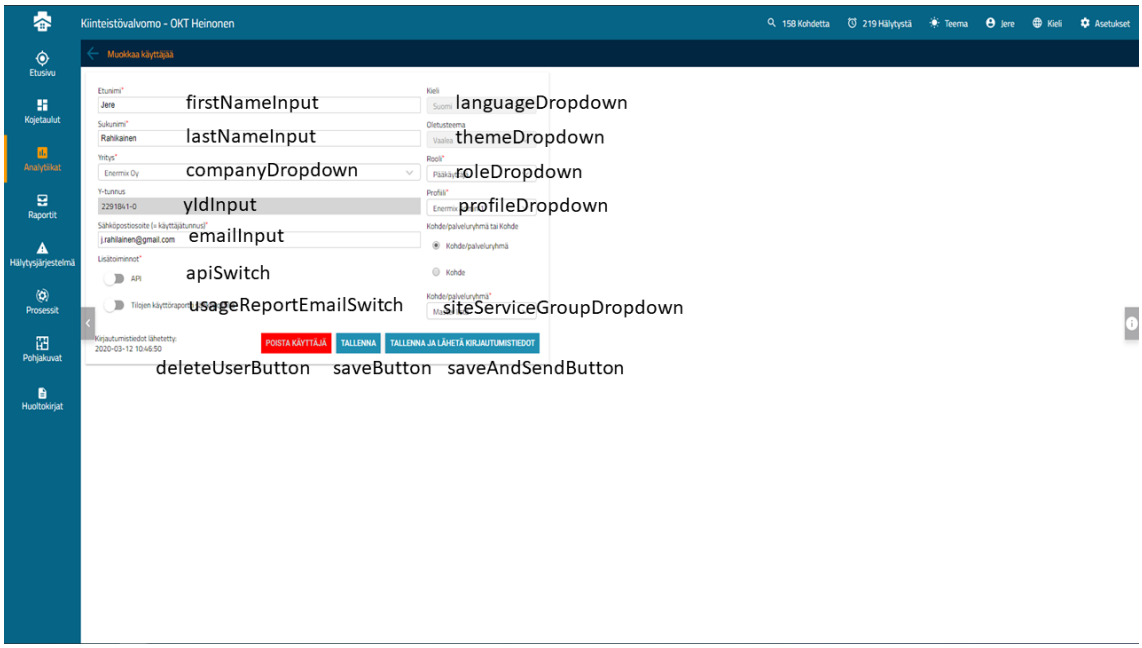
Robot Frameworkin testien nimeämiseen suositeltiin yleisesti Robot Frameworkin ohjeistusta. Ohjeistuksessa suositellaan nimeämään testitapaukset mahdollisimman selkeästi ja helposti lähestyttäväksi. Hyvä esimerkki testitapauksen nimeämiseen on esitelty tässä dokumentissa esimerkkikoodi 1:ssä. Testi on nimetty *Login - With proper credentials*.

Testien tunnisteiden nimeämiseen suositeltiin käytettäväksi Xrayn vaativaa asian tunnistetta sekä sen ohjelmiston versiota, jota testi koskee. Esimerkki testin tunnisteista seuraavassa koodiesimerkissä. Tämä kirjattiin README.txt tiedostoon, joka löytyy projektin juuresta.

```
[Tags] PRO-150  v2.1.0
```

Esimerkkikoodi 8. Robot Framework testien tunnisteiden nimeäminen.

Samalla kun ohjeistuksia luotiin niin kehitystä tapahtui myös automaatiotestien luomisessa. Uusia testejä syntyi muutamia ja samalla lisättiin ohjeistukseen Perspectivessä lisättävien elementtitunnisteiden nimeämisen ohjeistus, jotta testaaminen olisi vakaampaa tulevaisuudessa. Nimeämisessä pyrittiin erottamaan eri komponentit. Myös elementtitunnisteiden uudelleenkäyttö oli tärkeä määritelmä nimeämistyyliä valitessa. Esimerkiksi kaikkien tallenna nappien tunnisteeksi sovittiin SaveButton. Tämän avulla ei tarvitse luoda Robot Frameworkin puolelle kuin yksi viittaus tallenna nappiin. Kuvassa 17 on esimerkki elementtitunnisteille luodusta ohjeistuksesta.



Kuva 17. Ignition Perspective -elementtien tunnisteiden nimeämisen ohje.

Lisäksi ohjeistukseen liittyen käytiin suullisia keskusteluita ohjelmistonkehityksen parissa työskentelevälle henkilöstölle käyttäjätarinoiden merkityksestä testaamisessa ja ohjelmistokehityksessä. Samalla kertaa esiteltiin suullisesti ja live-esimerkin avulla Jiran alkuperäisten asiatyyppien liittämistä Xrayn tuomiin asiatyyppeihin.

5 Yhteenveto

Verraten lähtötilanteeseen yrityksellä on nyt paremmat valmiudet testikattavuuden kasvattamiseen hyödyntäen automatisoitua testausta manuaalisen testauksen rinnalla. Tämä mahdollistaa testauksen tehokkuuden sekä kattavuuden parantamisen. Toteutetut ohjeistukset ja uuden testausympäristön pystytys mahdollistaa testikattavuuden nostamisen. Testien tuottaminen on aloitettu ja Jiran käyttöä myös käyttäjätarinoiden osalta on hyödynnetty. Ohjeistuksia on tarkennettu uuden työntekijän koulutuksen yhteydessä. Kokonaisuutena yrityksen ohjelmistotuotannon kehitysmalli on uudistettu, se sisältää päivitetyn projektinhallinnan, kokonaan uuden testausympäristön ja automaatiotestauksen. Kuvassa 18 on havainnollistava kuva ohjelmistotuotannon kehitysmallista.

Kuva 18. Ohjelmistotuotannon kehitysmalli. Vanhaan malliin verrattuna testaus ja julkaisu on saanut lisää ominaisuuksia. (41.)

Kehitystä kohti automatisoitua työputkea on syytä jatkaa heti, kun testauksessa on saatu testikattavuutta kasvatettua. Automatisoitu työputki poistaa testien manuaalisen suorittamisen PyCharmissa sekä manuaalisen ajon palautteen lähettämisen Jiraan. Suositeltu työkalu Xrayn ja Jiran kanssa on esimerkiksi Jenkis, koska Xray tarjoaa valmiin lisäosan sen integroimiseen. Jenkins on suunniteltu automatisoimaan ohjelmistokehityksen osia, jotka liittyvät ohjelmiston kasaamiseen, käyttöönottoon ja testaukseen. Jenkins helpottaa jatkuvaa integrointia ja ohjelmistoversioiden toimitusta.

Yrityksen nostaessa automaatiotestaamisen päivittäiseen tekemiseen, saadaan kattavammin karsittua virheiden määrää julkaisuista ja siten loppukäyttäjäkokemus paremmaksi. Jiran hallitseminen nostaa tuottavuutta, kehitystiimi saa selkeämmän kuvan projektin suunnasta ja kehitettävästä kokonaisuudesta.

Lähteet

1 OPC Unified Architecture. Verkkoaineisto. Wikimedia Foundation. <https://en.wikipedia.org/wiki/OPC_Unified_Architecture> Päivitetty 25.04.2020. Luettu 03.05.2020.

2 MQTT. Verkkoaineisto. Mqttorg. <http://mqtt.org/> Luettu 03.05.2020.

3 Esineiden internet. Wikimedia Foundation. <https://fi.wikipedia.org/wiki/Esineiden_internet> Päivitetty 30.09.2019. Luettu 07.05.2020.

4 Talotohtori 2.0 – markkinoiden edelläkävijä. Verkkoaineisto. Enermix Oy. <https://www.talotohtori.fi/esittely> Luettu 06.05.2020.

5 Yksikkötestaaminen. Verkkoaineisto. Wikimedia Foundation. <https://fi.wikipedia.org/wiki/Yksikk%C3%B6testaaminen> Päivitetty 13.10.2019. Luettu 06.05.2020.

6 Kanban. Verkkoaineisto. Wikimedia Foundation. <https://fi.wikipedia.org/wiki/Kanban> Päivitetty 23.02.2020. Luettu 06.05.2020.

7 Deskopt Operating System Market Share Worldwide. 2020. Verkkoaineisto. statcounter GlobalStats. <https://gs.statcounter.com/os-market-share/desktop/worldwide> Päivitetty 06.05.2020. Luettu 07.05.2020.

8 What is Ignition. 2020. Verkkoaineisto. Inductive university. <https://www.inductiveuniversity.com/videos/what-is-ignition/8.0> Luettu 27.4.2020.

9 Fully Integrated Modules. 2020. Verkkoaineisto. Inductive Automation. <https://inductiveautomation.com/ignition/modules> Luettu 07.05.2020.

10 Perspective project elements. 2020. Verkkoaineisto. Inductive university. <https://www.inductiveuniversity.com/videos/perspective-project-elements/8.0> Luettu 27.04.2020.

11 Janne Heinonen. Talotohtorin arkkitehtuuri. 2019. Yrityksen sisäinen dokumentti. Enermix Oy.

12 BeginnersGuide/Overview. Verkkoaineisto. PyPA. <https://wiki.python.org/moin/BeginnersGuide/Overview> Päivitetty 18.09.2019. Luettu 07.05.2020.

13 Tool recommendations. Verkkoaineisto. PyPA. <https://packaging.python.org/guides/tool-recommendations/> Päivitetty 07.05.2020. Luettu 07.05.2020.

14 Examples. Verkkoaineisto. Robot Framework Foundation. <https://robotframework.org/#examples> Luettu 28.04.2020.

15 Browser drivers. 2020. Verkkoaineisto. GitHub, Inc. <https://github.com/robotframework/SeleniumLibrary#browser-drivers> Luettu 28.04.2020.

16 SeleniumLibrary. 2020. Verkkoaineisto. Libdoc. <https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html> Luettu 28.04.2020.

17 PyCharm. Verkkoaineisto. Wikimedia Foundation. <https://en.wikipedia.org/wiki/PyCharm> Päivitetty 21.04.2020. Luettu 28.04.2020.

18 Versionhallinta. Verkkoaineisto. Wikimedia Foundation. <https://fi.wikipedia.org/wiki/Versiohallinta> Päivitetty 06.11.2019. Luettu 29.04.2020.

19 About. Verkkoaineisto. Git. <https://git-scm.com/about> Luettu 01.05.2020.

20 Small and Fast. Verkkoaineisto. Git. <https://git-scm.com/about/small-and-fast> Luettu 01.05.2020.

21 Jira for task management. 2020. Verkkoaineisto. Atlassian. <https://www.atlassian.com//jira/guides/use-cases/what-is-jira-used-for#jira-for-task-management> Luettu 05.05.2020.

22 Ketterä ohjelmistokehitys. Verkkoaineisto. Wikimedia Foundation. <https://fi.wikipedia.org/wiki/Ketter%C3%A4_ohjelmistokehitys> Luettu 05.05.2020.

23 Max Rehkopff. Agile Epics: Definition, Examples, & Templates. 2020. Verkkoaineisto. Atlassian. <https://www.atlassian.com/agile/project-management/epics> Luettu 05.05.2020.

24 Max Rehkopff. User Stories With Examples and Template. 2020. Verkkoaineisto. Atlassian. <https://www.atlassian.com/agile/project-management/user-stories> Luettu 05.05.2020.

25 Bruno Conde. About Xray. Verkkoaineisto. Xpand IT. <https://confluence.xpandit.com/display/XRAYCLOUD/About+Xray> Päivitetty 11.07.2019. Luettu 05.05.2020.

26 The Ultimate Guide: Test Management for Atlassian Jira. 2019. Verkkoaineisto. Eficode Root. <https://www.eficoderoot.com/blog/the-ultimate-guide-test-management-for-atlassian-jira> Luettu 07.05.2020.

27 Automation Testing Vs. Manual Testing: What´s the Difference. Verkkoaineisto. guru99. <https://www.guru99.com/difference-automated-vs-manual-testing.html> Luettu 17.04.2020.

28 Unit Testing. Verkkoaineisto. Wikimedia Foundation. <https://en.wikipedia.org/wiki/Unit_testing> Päivitetty 04.05.2020. Luettu 07.05.2020.

29 Sten Pittet. The Different types of Tests. Atlassian. <https://www.atlassian.com/continuous-delivery/-testing/types-of--testing> Luettu 07.05.2020.

30 cmallonee. 2020. Jump from the top of the page to bottom. Verkkoaineisto. Inductive automation. <https://forum.inductiveautomation.com/t/jump-from-the-top-of-the-page-to-the-bottom/32154> Päivitetty 12.01.2020. Luettu 08.05.2020.

31 Python 3.8.2. 2020. Verkkoaineisto. python. <https://www.python.org/downloads/release/python-382/> Luettu 08.05.2020.

32 Downloads. 2020. Verkkoaineisto. Git. <https://git-scm.com/downloads> Luettu 08.05.2020.

33 Thank you for downloading PyCharm. 2020. Verkkoaineisto. JetBrains. <https://www.jetbrains.com/pycharm/download/download-thanks.html?platform=windows&code=PCC> Luettu 08.05.2020.

34 Installation instructions. Verkkoaineisto. GitHub. < https://github.com/robotframework/robotframework/blob/master/INSTALL.rst> Päivitetty 13.09.2018. Luettu 08.05.2020.

35 Selenium2Library. Verkkoaineisto. GitHub. <https://github.com/robotframework/Selenium2Library> Päivitetty 04.12.2017. Luettu 08.05.2020.

36 Bruno Conde. Manual provisioning. Verkkoaineisto. Xpand IT. <https://confluence.xpand-it.com/display/XRAYCLOUD/Using+Generic+Tests+for+Automation> Päivitetty 16.08.2018. Luettu 07.05.2020.

37 Bruno Conde. Installation. 2018. Verkkoaineisto. Xpand IT. <https://confluence.xpand-it.com/display/XRAYCLOUD/Installation> Päivitetty 30.10.2018. Luettu 08.05.2020.

38 Bruno Conde. Quick Setup. 2019. Verkkoaineisto. Xpand IT. <https://confluence.xpand-it.com/display/XRAYCLOUD/Quick+Setup#QuickSetup> Päivitetty 10.09.2019. Luettu 08.05.2020.

39 Sergio Freire. Global Settings: API Keys. Verkkoaineisto. <https://confluence.xpand-it.com/display/XRAYCLOUD/Global+Settings%3A+API+Keys> Päivitetty 18.06.2018. Luettu 08.05.2020.

40 Sergio Freire. Authentication – REST. Verkkoaineisto. <https://confluence.xpand-it.com/display/XRAYCLOUD/Authentication+-+REST> Päivitetty 20.02.2019. Luettu 08.05.2020.

41 Janne Heinonen. Palveluhallintamalli. 2020. Yrityksen sisäinen dokumentti. Enermix Oy.